EVOLVING COMPLEX VISUAL BEHAVIOURS USING GENETIC PROGRAMMING AND SHAPING

SIMON PERKINS

Los Alamos National Laboratory, MS D436, Los Alamos, NM 87544, USA
E-mail: s.perkins@lanl.gov

GILLIAN HAYES

Division of Informatics, University of Edinburgh, 5 Forrest Hill, Edinburgh, Scotland
E-mail: gmh@dai.ed.ac.uk

Shaping is a way in which a human designer can provide assistance to a learning system to enable it to solve problems that would otherwise defeat it. The experiments reported here explore a variety of shaping techniques used in conjunction with a genetic programming based system, in order to develop controllers for visual tracking tasks. Controllers are evolved in simulation using shaping, and are then transferred successfully to a real robot head.

1 Introduction

1.1 From Robot Learning...

For many years now, automatic design approaches based on techniques from Genetic Algorithms, Reinforcement Learning and Neural Networks, have been touted as the way in which the engineers of the future will produce robot control systems. Just specify at a high level what you want the robot to do, and then sit back and wait while the GA/RL/NN works out how to get the robot to do it. While this is undoubtedly a very attractive goal, a quick look at the current state of the art of automated robot design reveals that in most cases we are a long way from being able to automatically design controllers that can out-perform human-designed ones. While robot controllers have been learned from scratch for very simple 'insect-like' behaviours, general-purpose *tabula rasa* learning of complex tasks seems difficult or impossible.

1.2 ... To Robot Shaping

One particularly promising solution to this problem is to bring the human back into the design cycle: rather than attempt to learn a complex task in one go, learn it in an incremental, hierarchical fashion, with the human providing a task-specific framework, and the learning system 'filling in the blanks'. While

learning the whole task may be beyond the capabilities of the learning system, learning small parts of it and then learning how to put these parts together may not be. There are many examples of work using this idea for robot controller development.[1,2,3] The approach has been called 'robot shaping',[4] based on the usage of the word in the psychology of animal training.[5]

As we shall see, a great variety of shaping techniques can be applied to robot learning, and they can be combined with a great variety of different learning architectures. What shaping methods should we use? And what learning architectures should we use with them? The remainder of this chapter examines a number of choices and applies them to the difficult problem of learning a visual tracking behaviour for a robot head.

2 Task Domain

2.1 Visual Tracking

If we are going to demonstrate the usefulness of shaping, then we need a reasonably complex task domain. Tasks involving vision are notoriously hard for learning systems due to the vast quantity of data that video cameras produce each second, coupled with the relatively high difficulty in interpreting visual pixel data, compared with, for instance, sonar echoes. As a result they make a good testbed for shaping techniques.

Tracking moving objects using vision is a basic skill in most visually equipped animals and has been widely studied in the robotic vision community. A robotic 'camera head' that can fixate on moving objects is a commonplace sight in robotics labs around the world, but virtually all of these have been carefully programmed by hand. Our experiments are directed at getting a robot to learn a controller that can reliably perform this task, with little or no pre-processing of the visual data.

As a milestone on the way to the goal of tracking arbitrary moving objects, we have performed experiments on the easier task of 'light-tracking', in which the moving target to be tracked is distinguished by being significantly brighter than its surroundings.

Our robotic platform consists of an RWI B21 mobile robot equipped with a single colour video camera mounted on a Directed Perception pan/tilt unit (PTU). For light-tracking, the behaviour we require is that the robot should keep the brightest object in front of it centred in its visual field. For the harder problem of motion-tracking, we require that the arbitrarily coloured moving object in the scene be kept centered in the image.

Learning algorithms typically take a long time to run. This problem is often magnified for robots since the speed at which controllers can be evaluated is limited by the time it takes the robot to physically move. Therefore in common with many other researchers,[6,7] we have decided to train the controller in simulation initially and then attempt to transfer it directly to the real robot. Using simulations has an additional advantage: it allows us to use more 'informed' evaluation functions that can make learning much faster.

Simulations are often criticized on the grounds that it is very unlikely that a controller evolved to work in a simulator will ever work anywhere else due to the impossibility of accurately modeling the real world. However, several researchers[7,8] have shown that with a little care, it is possible to perform this simulation to reality transfer.

At first thought, it might seem that the 'best' simulator possible is the one that most closely models the real world. Perversely however, it turns out that in fact you can do better by making the simulation worse. Jakobi[7] calls this the 'radical envelope of noise hypothesis' and it has three main tenets: (1) For features of the real world that are relevant to the behaviour, model them as closely as possible and then add a bit of noise to account for the modeling inaccuracy; (2) For features in the real world that are not relevant to the behaviour, model them poorly and add lots of noise to prevent the learning system using those features; (3) Don't bother to model eventualities which are not part of the target behaviour since they won't be encountered when that behaviour has been learned correctly.

One problem with this approach is that it requires the designer to decide in advance which parts of the environment to model accurately, and which to leave out, and it is often difficult to do this without having at least some idea how the robot is going to solve the problem being set. So this strategy does put some additional demands on the designer, but the hope is still that for complicated tasks, the advantages in using simulations outweigh the additional design complexity.

Following from this, our simulator doesn't attempt to model a realistic scene — instead, the image produced by the simulator consists of a randomly textured background, containing a fairly crudely modeled 'diamond-shaped' target. For the light-tracking task, the target is significantly brighter than the background. For the motion-tracking task, the target is randomly coloured in the same fashion as the background. An important feature of the simulator derives from the observation that real backgrounds are not uniformly random, but contain both small and large scale variations. To attempt to model this,

the texture of the random background is modulated by a coarse scale 'mosaic' scene. Figure 1 shows a robot's-eye view illustrating these features for both tasks.

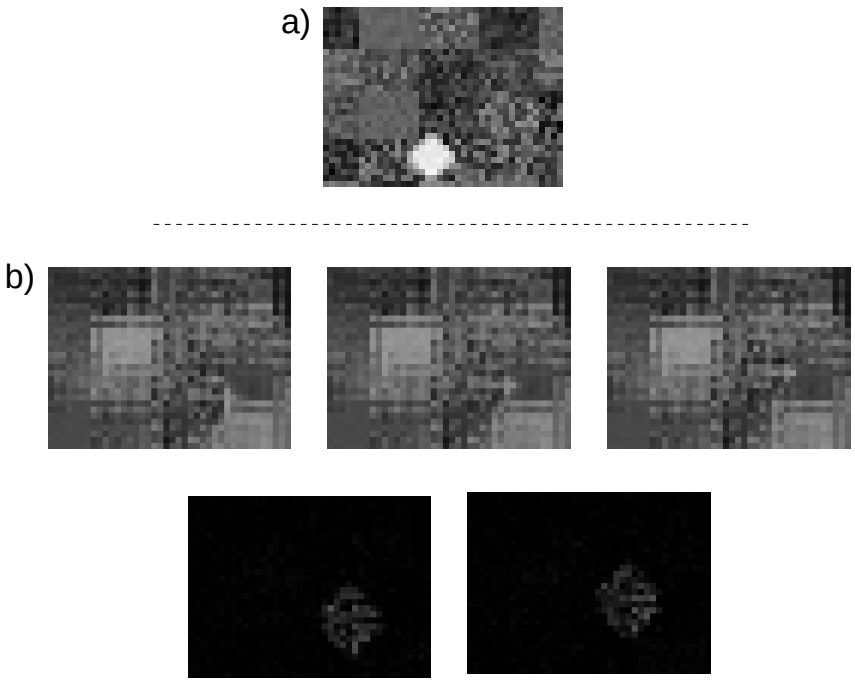

Figure 1. (a) Typical simulator image for the light-tracking task. (b) Typical simulator image sequence for the motion-tracking task. The top row of images shows a sequence of three frames produced by the simulator with the camera stationary. The moving target is very difficult to spot in these images, but can be seen more clearly in the bottom row of images, which shows the absolute difference between images 1 and 2, and between images 2 and 3.

Note that it is not necessary to model the background scene in a realistic way. However, it *is* necessary to model the way in which the target moves fairly accurately. This was achieved by basing the imaging model on empirical measurements of known scenes taken using the real robot head. Building a simulator using empirical data has previously been suggested by Miglino.[8] On the actuator side, the stepper motors on the robot head actually allow almost error-free positioning, but the velocity signals sent to the simulator

by the controller are corrupted by adding multiplicative Gaussian noise with $\mu = 1.0, \sigma = 0.02$ to mask image modelling inaccuracies.

3 The Elements of Shaping

3.1 Learning Algorithms

There are a number of learning systems which could be used in shaping framework, but we have decided to opt for a genetic algorithm.[9] There are several reasons for this.

Firstly, complex robotic tasks often fall into a 'reinforcement learning' framework, where the only feedback available is some measure (typically scalar) of how well the robot is doing, rather than a 'supervised learning' framework, where the trainer knows exactly what the robot should be doing in a number of example cases. This largely precludes techniques such as classic neural network gradient descent.[a]

Secondly, we decided from the start that we wanted to deal with continuous state and action spaces. This makes using classic 'reinforcement learning' techniques involving value functions, such as Q-learning,[11] problematic, although there are adaptations of the basic methods that can work in such continuous spaces.[12,13]

Genetic algorithms however are well suited to working in continuous spaces and with scalar evaluation. A crucial issue that must be decided is how to represent candidate robot controllers so that they can be manipulated effectively by the GA. Our system is based up 'genetic programming'[14] (GP) and is called TAG.[15]

The TAG *Architecture*

TAG extends basic GP in a number of ways. The basics of GA/GP are not covered here. Consult a suitable reference[16,17] for details.

In contrast to standard GP, the structures evolved by TAG are general acyclic graphs rather than trees. Evolving graphs rather than trees allows controllers that use values computed by sub-graphs more than once to be be expressed in a more compact and evolvable way than is possible with simple trees. Several other authors have suggested adding graphs into GP for similar reasons.[18,19] TAG graphs are initially created by constructing a 'bag' of randomly chosen GP-style function and terminal nodes. The number

[a] Although see Complementary Reinforcement Back Prop[10] for something similar to gradient descent using reinforcement signals and binary neural networks.

of nodes in the bag is chosen randomly within a certain range: 10–20 nodes per bag in these experiments. TAG graphs can have more than one output value (for instance one output per actuator), and for each required output, a node is selected from the bag at random to return that value. If the node is not a terminal node, then it requires input values on which to perform its calculation, so other nodes are selected from the bag and connected to the first node as needed. The process continues in recursive fashion until there are no nodes in the graph that need input connections. Loops are prevented by insisting that no node may connect to a node that is its ancestor (i.e. its parent, or an ancestor of its parent), and in order to ensure that the process terminates it is necessary that the bag contains at least one terminal node to start with. At the end of the connection process, not all nodes are necessarily in use — these unused nodes act as spare genetic material or 'introns' that may be used later. Figure 2 shows a typical TAG graph.

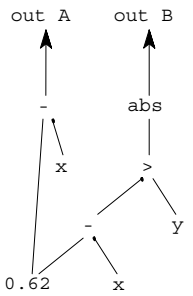

Figure 2. A typical small graph evolved using TAG.

Traditional GP uses sub-tree crossover as its primary genetic operator. TAG also uses crossover, but the operator must be modified for use with graphs. For the most part, 'sub-graph crossover' is very similar to sub-tree crossover. Starting with two parents, two offspring graphs are created by copying. Then, a node is selected in each offspring to act as an exchange node. To perform crossover, the exchange nodes are simply swapped between the two graphs, together with all the descendent nodes of the exchange node (a node is a descendent of another node if it is a child of that node or a descendent of a child).

The tricky part of sub-graph crossover is deciding what to do about connections that previously connected into the exchanged sub-graphs (other than connections to the exchange node themselves). One obvious answer is just to randomly reassign such connections, but TAG tries to take a more intelligent

approach that attempts to preserve structure where possible. In brief, every node is associated with a fixed 'tag' value that is uniquely assigned when that node is first created. When a node is copied, the copy is given the same tag value. If a particular sub-graph turns out to be a useful component, then copies of it will tend to multiply in the population. Each of those copies will have similar sets of nodes and associated tags. When TAG is reassigning a connection into an exchanged region of nodes, it first checks to see if there are any nodes present in the newly formed individual that have the same tag as the node that the connection was previously connected to. If there is, then a connection is made to that node. If not, then the connection is reassigned randomly. The key idea is that it is less destructive if connections are reassigned to structures that are similar to the ones they were previously connected to. Figure 3 illustrates the general idea.

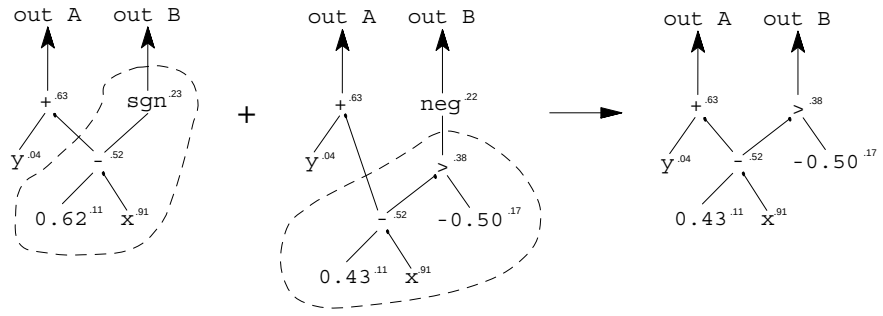

Figure 3. An example of crossover in action. Only one offspring node is shown for clarity. The numeric labels next to each function node show the associated tag values. During crossover, the subgraph rooted at the node with tag 0.23 in left parent is deleted, and replaced by the subgraph rooted at the node with tag 0.38 in the right parent. In the child, the target value of the `out B` output node, which originally pointed to the root node of the deleted subgraph, as been changed so as to target the root node of the inserted subgraph. Notice also that connection from the plus sign, which pointed at a deleted node, has automatically reassigned itself to a subgraph rooted at a node with an identical tag value to the deleted node. As a result, the subgraph that calculates the values of `out A` has changed only very slightly (the constant 0.62 has changed to 0.43). The new subgraph that the connection points to is probably a mutated version of the original subgraph.

Offspring can also be produced by mutation. In this case, a single child is initially generated by copying a single parent. Then, R nodes are selected at random for mutation, where R is Poisson-distributed with expected value 2.[b] Some types of node have internal parameters that are 'micro-mutatable',

[b]If $R = 0$, then R is re-generated.

and if such a node is selected, then 90% of the time, one of its parameters is mutated slightly. In all other cases the node has one of its input connections reassigned randomly (if applicable), or is itself replaced with a random node.

TAG has a number of other interesting features, principally the use of a 'rational allocation of trials' (RAT) mechanism[20] for reducing fitness evaluation time, but space precludes a fuller description here. See Perkins[15] for details.

Apart from the aforementioned exceptions, TAG is a relatively conventional evolutionary algorithm. It is a 'steady-state' algorithm in that as soon as offspring are generated they are put back into the evolving population — there is no concept of a generation. TAG maintains a population of size N. Parents are selected using tournament selection: at each evolutionary step, M individuals are chosen from the population at random and their fitnesses are compared. The fittest individual in the tournament is chosen as one parent, and one of the remaining individuals is chosen randomly as the other parent. 50% of the time the two parents are bred using crossover. Only one of the two potential children is actually generated. The other 50% of the time, mutation is used to derive a single offspring from the fitter of the two parents. In either case the offspring replaces the less fit of its parents. This evolutionary cycle is repeated a total of T times during a single run. In the experiments reported here, $N = 100$, $M = 4$ and usually, $T = 25000$.

3.2 Shaping Methods

There are many ways in which a human expert with knowledge about a particular task can assist a robot learning system. Here we concentrate on just two:

Controller decomposition Controllers for complex tasks can often be broken down into a hierarchy of smaller modules, in a manner analogous to the way in which human programmers structure a program into functions and objects. It is often much easier to train these individual modules separately or sequentially, than to train the whole controller at once. In robotics, controllers are often decomposed in a behaviour-based way, with some modules performing simple sub-tasks, and others coordinating the activation of those modules. Mahadevan *et al.*[21] used this approach, training a robot to push boxes by hand-designing the coordination modules, but learning the primitive sub-tasks. Dorigo[2] went one step further and trained both the primitive sub-tasks and the coordination behaviours.

Progressive problem difficulty One important problem faced by learning

robots is the difficulty of 'getting off the ground'. At the start of the learning process the robot will probably behave in a relatively random fashion, and in some training scenarios this might mean that the robot gets very little useful feedback on how to behave correctly. In a delayed reward scenario, for instance, the robot might never reach the rewarded goal by chance and hence learning will never happen. This problem can sometimes be alleviated by initially training the robot on easier versions of the full task. Asada[22] calls this 'learning from easy missions', and they show that the process can significantly accelerate learning.

A fuller taxonomy of shaping methods can be found in Perkins.[15,23] For more discussion of robot shaping in general see Dorigo and Colombetti.[4]

3.3 Incremental Learning

The shaping approach implies an incremental acquisition of behaviour with new learned skills building upon previously acquiring skills. The human trainer must design an incremental path of increasing competence that the robot is to follow. Each 'stage' along this path or 'shaping regime', results in the acquisition of another unit of competence in some area related to the overall task. At each shaping stage, the task facing the robot is likely to be slightly different, and the schedule for receiving rewards will in general be changed as well. The final stage of a shaping regime usually requires the robot to perform the full task without special assistance.

Training a robot using either of the above shaping methods requires a learning architecture that is capable of learning incrementally. The controller must not forget skills it has already learned, even if they are not directly relevant to the current stage of the training process, and it must be able to combine previously learned skills together to form new, more complex skills.

We have explored three different frameworks for incremental learning:

Single-Agent (SA) In the simplest case, no special additions are needed to the basic evolutionary learning mechanism. The controller for the robot consists of a single TAG graph, and a single population of evolving individuals is maintained as the robot moves through the shaping regime. The hope is that as the shaping environment changes, the evolving controller population will adapt to those changes as they occur. This technique is only suitable for shaping regimes where the task faced by the robot does not change qualitatively from one stage to the next.

Multiple-Agent, Fixed-Interaction (MAFI) The alternative to a single agent controller is a multi-agent controller. In this case, the final con-

troller consists of an interacting collection of TAG graphs, each of which is referred to as an 'agent'. During the initial evolutionary phase, the controller consists of a single agent. At the end of that stage, the fittest agent in the evolving population is 'frozen' and incorporated permanently into the controller. A new population is then created for the second stage and a new agent is evolved that works with the first. The process can be continued indefinitely, with each evolutionary stage adding one more agent to the controller. In the MAFI framework, each agent has a separate independent role within the controller and either they act without conflict with one another, or there exists an externally supplied arbitration system that decides which agent controls the robot in the case of conflict.

Multiple-Agent, Learned-Interaction (MALI) Finally, we can consider a framework identical to MAFI, except that the arbitration between agents is be learned rather than designed by a human expert. There are several possible ways of doing this, but we use a system we call 'hierarchical evolutionary gating' (HEG). Agents are evolved as before, one per evolutionary stage. This time though, each agent is given an additional output (in addition to the usual actuator outputs) producing a value called the 'validity'. The validity is used to determine whether an agent 'believes' itself to be confident of proposing a correct action at the current time. If the validity is greater than or equal to zero the agent is eligible to control the robot. If it is less than zero, the agent is ineligible. For each actuator over which there is conflict between agents, we only consider those agents with positive validities. If there is still conflict, then the most recently evolved agent wins control. If no agents are eligible, the actuator assumes a default value for the current cycle. The validity mechanism allows an agent to take control if it needs to override previously evolved agents, and to relinquish control to previously evolved agents if that is not necessary. Note that since different agents can have different sets of outputs, an agent which is overridden with respect to actuator A, may get to control actuator B.

In general a shaping regime can use different learning frameworks for different stages of the shaping regime.

4 Experiments and Results

4.1 Experimental Setup

Sensors and Actuators

A successful controller has to learn how to link sensing to action in a way that performs the desired behaviour. On the sensing side, our controllers are provided with access to the image data from the camera at a simulated 4Hz frame rate. The camera image contains $320 \times 240 = 76800$ pixels. Before being presented to the controller, pixel intensities are converted into real numbers ranging from 0.0 (black) to 1.0 (white).

On the action side of the controller sits the PTU, which in our setup is 'speed-controlled', with independent control of pan and tilt axes. The controller outputs two real numbers between 0.0 and 1.0, one for the pan axis, one for the tilt axis. 0.0 is interpreted at maximum speed left/down, 1.0 is interpreted as maximum speed right/up and 0.5 is interpreted as stationary.

Functions and Terminals

A crucial task specific aspect of using TAG is the choice of function and terminal nodes that graphs may be made out of. The function nodes used in TAG are largely similar to the ones used in standard GP, consisting of the standard arithmetic functions: $+$, $-$, $\times$, $\div$, $>$, `neg`, `sgn` and `abs`; plus the ephemeral random constant $\Re$. Division ($\div$) returns 1.0 if its second argument is zero. Greater-than ($>$) returns 1.0 if its first argument is greater than its second argument, or 0.0 otherwise. `sgn` returns 1.0 if its single argument is greater than zero, or -1.0 otherwise. Random constants are initialized to values between -1.0 and 1.0.

In addition to these familiar functions two additional terminal nodes and three additional functions are introduced for the tracking tasks. `vis` provides access to the camera sensor. It contains five internal parameters that define a rectangular 'receptive field' in the image. Four parameters define this rectangle, and the fifth defines a sampling resolution within that receptive field. `vis` is unusual in that it returns a 2-D *array* of values. The arithmetic functions described above are designed to cope with arrays in a sensible fashion. If one input is scalar and the other is an array, then the output is another array of the same size as the input array, in which each value is the result of applying the function to the scalar value and the corresponding value in the input array. If both inputs to a function are arrays then the arrays are first aligned so that as near as possible their centres are co-registered. Then the

largest common sub-array is found and an array of that size is returned in which each value is computed from the corresponding values in the two input arrays. The `vis` terminal is only used for the light-tracking task. For the motion tracking task, a slightly different terminal `vis2` is used, which returns an array of values representing the absolute *difference* in intensity values in a specified receptive field between the current frame and the previous one.

In conjunction with this terminal node, three simple image processing function nodes are defined: `avg`, which returns the mean value of an array; `mx`, which returns the first x-moment of the array, and `my`, which returns the first y-moment of the array. These functions simply return their input if passed scalar values.

Fitness Evaluation

The basic fitness function for the tracking tasks is defined in terms of the tracking error E, i.e. the absolute angular offset between the straight-ahead direction of the camera and the centre of the target. Given this, the fitness is:

$$F = -\sum_{t}^{n}(1-\gamma^t)E_t \tag{1}$$

where E_t is the tracking error at time-step t, γ is a weighting constant, set to 0.5 in these experiments, and the trial is continued for n time-steps — in these experiments $n = 10$. Note that the sum is negated to ensure that higher fitness values are better.

Simulation Details

In order to accurately assess the fitness of controllers it is necessary to test them in a number of different simulator situations, known as fitness cases. Each fitness cases consists of a particular combination of target size, target speed, target starting location and background texture. These are generated pseudo-randomly, and each controller is tested on between 10 and 50 different fitness cases, the exact number being decided dynamically using the RAT mechanism mentioned briefly above.

For the light-tracking task, pixels in the simulated background vary in intensity value between 0 and 150. The diamond shaped simulated target varies in radius between 4° and 10°, and in intensity between 230 and 240. The target moves in the simulated scene with a velocity between 0 and $10°s^{-1}$, and the simulator runs on a 4Hz cycle.

For the motion-tracking task the target is coloured in the same random way as the background. Both background and target pixels take values between 0 and 240, and the target moves with a velocity between 5 and $20^{\circ}s^{-1}$. Other details are the same as for light-tracking.

4.2 Progressive Problem Difficulty Experiments

The first set of experiments looked at whether the shaping technique of progressive problem difficulty could be successfully applied to learning a 1-D version of the light-tracking task, in which only the pan motor of the robot head need be controlled.

A series of two stage shaping regimes was explored. In each case the tracking problem was in some way simplified in the first stage in order to assist the learning mechanism, and then that simplification was removed in the second stage. Both the SA and MALI frameworks for incremental learning were tested. For each framework, a control experiment with no shaping was also performed. In all, ten experiments were carried out:

(a) Control (MALI).

(b) Control (SA)

(c) Restricted target start position: for the 1st stage only, targets are restricted to appearing in the left half of the image (MALI).

(d) Restricted target start position (SA).

(e) Stationary targets: for the 1st stage only targets are stationary (MALI).

(f) Stationary targets (SA).

(g) Large targets: for the 1st stage only, targets are always 10° in radius (MALI).

(h) Large targets (SA).

(i) Increased contrast: for the 1st stage only, the background intensity values are reduced to half normal values (MALI).

(j) Increased contrast (SA).

Results

For each experiment, 50 independent runs with different random number seeds were carried out. Figure 4 shows the performance graphs from all 10 experiments. The solid line in each graph shows the median (over 50 runs) minimum tracking error in the population *vs.* number of tournaments. The dotted lines show the 10th and 90th percentile minimum errors over the 50 runs.

The first thing to note about the final results is that they are all rather similar. Experiment (c) has a median minimum error that is marginally significantly better that the control ($p = 0.05$ using a two-tailed randomization-based statistical test[24]), and experiment (d) has a median minimum error that is significantly worse than the control ($p < 0.01$), but all the other experiments did not produce median results that differed significantly from their corresponding control.

The experiments do however differ in how the population approaches the final result. Many of the graphs in Figure 4 have a discontinuity at the halfway point which marks the point where the first stage ends and the second stage begins. The sudden increase in minimum error at this point in many of the graphs, followed by a gradual decrease in error again, is due to the shaping conditions being changed at this point which forces the population to undergo a period of re-adaptation to the new conditions.

As expected, the error at the end of the first (simplified) stage in most of the shaping runs is lower than the error at the corresponding time in the control runs, and in many of the runs is lower than the final error achieved. Experiments (i) and (j) show this quite clearly — the increased target contrast in these experiments makes the tracking problem easier to solve, and so the evolved controllers have a lower error than usual. Unfortunately this does not translate into finding solutions with significantly lower errors than usual on the unsimplified task. It seems that making the problem easier has not allowed TAG to find *better* solutions than in the control experiments. Instead, TAG finds rather similar solutions that happen to perform better on the unsimplified task.

The experiments using restricted target start positions (RTSP) are in many ways the most interesting results here. Using the MALI framework, we obtain results that are significantly better than the control. Using the same shaping regime, but in conjunction with the SA method, we obtain results that are significantly worse than the control. This discrepancy appears to be evidence of agent specialization at work. During the first stage under either framework, the single agent learns to deal exclusively with targets appearing on the left side of the image. Judging by the graphs this is easier than learn-

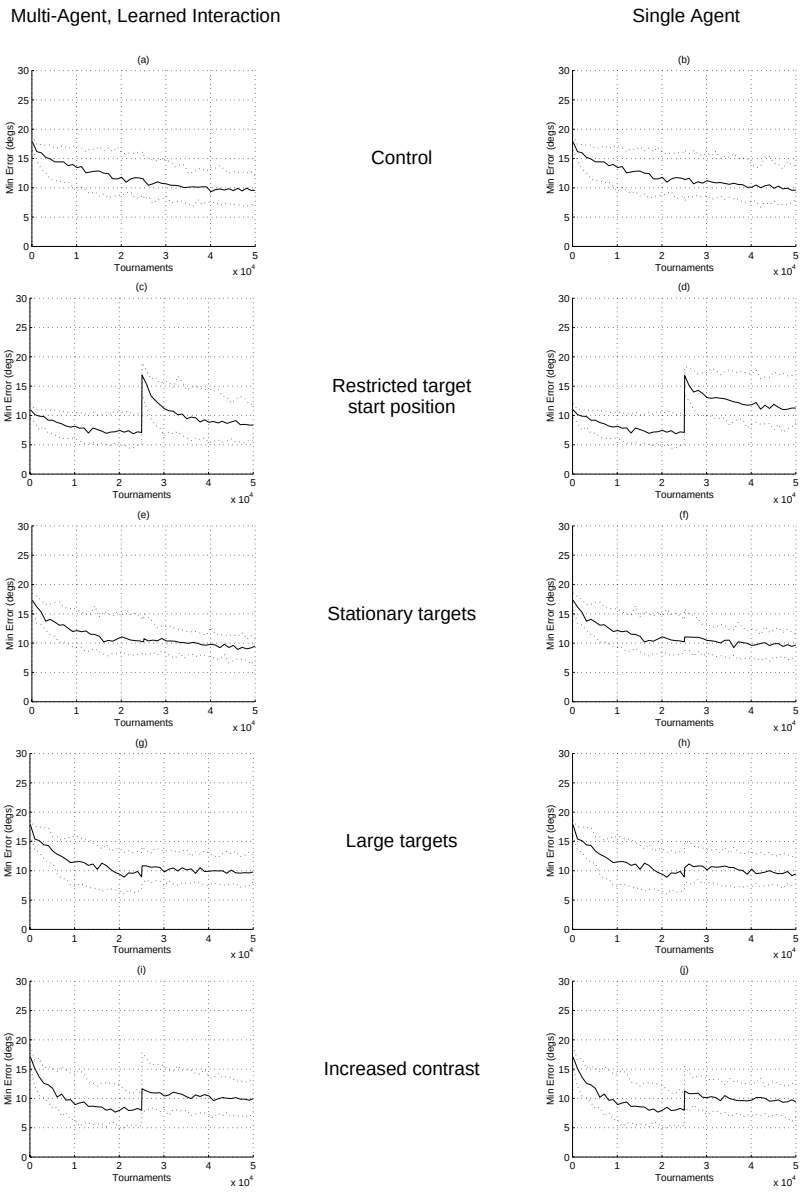

Figure 4. Progressive problem difficulty and the 1-D light-tracking task.

ing to deal with the whole image. During the second stage, under the MALI framework the second agent learns to effectively track targets appearing on the right side of the image and simultaneously learns to set the validity appropriately so that each agent is active at the correct time. This is possible because the RTSP shaping regime matches a division of labour that is relatively easy for TAG to discover. In contrast, under the SA framework, the population has already specialized to only deal with targets on the left side by the end of the first stage. This specialization makes it difficult for the population to then learn how to deal with targets appearing on the right side of the image.

The relative failure of the other shaping regimes seems to come down to the shaping regime not corresponding to a division of labour which is easy to discover by TAG.

4.3 Controller Decomposition Experiments

The second set of experiments explored the idea of using controller decomposition to simplify learning the same 1-D light-tracking task. The idea here is that rather than trying to learn the tracking task in one go, it would be easier to learn how to *locate* the light accurately first of all, and only then to learn how to track it.

For the light-location stages, a different fitness evaluation metric is needed. TAG graphs evolved during these stages have a single output whose value is written to a memory cell. The value in this cell is compared with the horizontal coordinate of the known location of the centre of the target in the simulated scene and the assigned fitness is then simply the negation of this location error. Note that the action of writing to the memory cell is treated as the same as writing to an actuator for the purposes of conflict resolution.

In addition to the controller decomposition, progressive problem difficulty using the restricted target start position regime was employed to assist learning the light-location stages.

A five stage shaping regime was used:

1. 1-D light-location task with targets appearing only in the leftmost 25% of the image.

2. 1-D light-location task with targets appearing only in the leftmost 50% of the image. The first agent is frozen and a second evolving agent added under the MALI framework.

3. 1-D light-location task with targets appearing only in the leftmost 75% of

the image. The second agent is frozen and a third evolving agent added under the MALI framework.

4. 1-D light-location task with targets appearing anywhere in the image. The third agent is frozen and a fourth evolving agent added under the MALI framework.

5. 1-D Light-tracking task with targets appearing anywhere in the image. The fourth agent is frozen and a fifth agent is added under the MAFI framework. The new agent has access to a new terminal node `mem0` which reads the memory cell written to by the first four agents.

Each stage is 25000 tournaments long. A control experiment was carried out in which a single agent controller was trained on the unsimplified 1-D light-tracking task over 125000 tournaments, with no shaping.

The different controller architectures that result from the two experiments are shown in Figure 5.

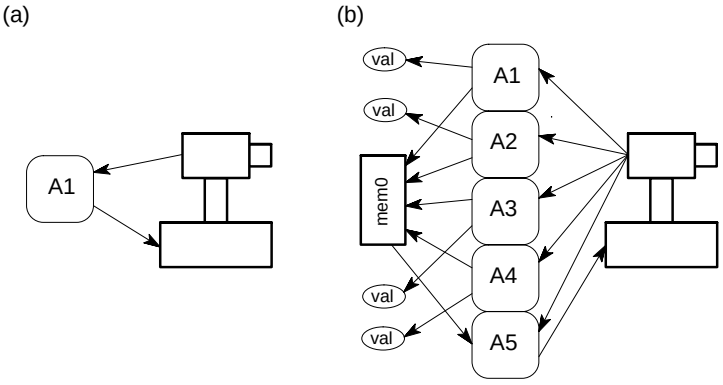

Figure 5. Controller architectures used in shaping experiments. (a) Single-agent control experiment. (b) Five-agent controller produced using progressive problem difficulty and controller decomposition.

Results

Figure 6 shows the performance graphs from these experiments. The solid line plots the median location/tracking error of the best individuals from each of the 50 runs of each experiment, plotted against number of tournaments. Note that the location/tracking error is simply the negative of the fitness metric

being used. Also shown are the 10th and 90th percentile lowest errors from the 50 runs. A two-tailed randomization test showed that the controllers produced by the shaped experiment have a significantly lower median error than those produced by the control ($p < 0.01$).

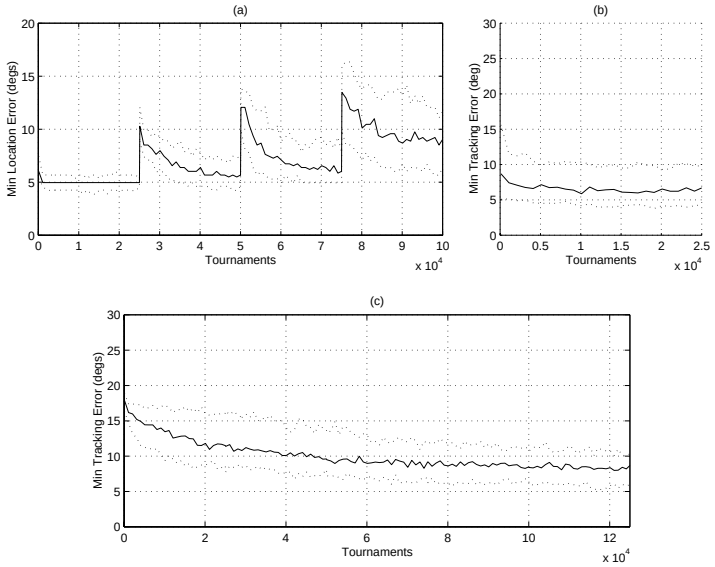

Figure 6. Performance graphs for the controller decomposition experiments. (a) Performance graph for stages 1–4 of the experiment. (b) Performance graph for stage 5 of experiment. (c) Performance graph for the control experiment.

4.4 Back to Reality

It is one things to show that shaping can help us produce better robot controllers in simulation, but another thing altogether to show that this leads to better controllers in reality. To test the ability of the evolved controllers to work on the real robot, further shaping runs were carried out to attempt to evolve controllers for the full 2-D light and motion tracking tasks that could compete with hand-designed controllers. Space precludes a detailed description of the experimental procedure here, but a full description can be found in Perkins.[15] In brief though, a ten stage shaping regime, similar to the one described in Section 4.3, was used to produce controllers that performed well

on the 2-D light and motion tracking tasks in simulation. These controllers were then transferred directly to the real robot head upon which the simulator was based, and tested. Handmade controllers for each of these tasks were also produced, and these were tested in the same way. The hand-designed controllers made use of the same sensor data as was available to the evolved controllers, and only made use of functionality that could be performed by a TAG graph. The light-tracker worked by thresholding the image to find the light target, determining the centroid of the target, and then moving the robot head accordingly. The motion-tracker worked in a similar way except that the target was found by only moving the robot head on alternate frames, and performing frame-differencing to locate motion.

The light-tracking controllers were tested by darkening the room, and then holding a desk lamp in front of the robot at a distance of about 1.5m from the camera. The lamp was held stationary in 100 different positions, and the controller was allowed to attempt to point the camera at the light. Performance was measured by using simple image processing to locate the centre of the light in the image, and determining the offset from the optical axis of the camera.

The motion-tracking controllers were tested using a toy racing car set. A simple track layout was created, and a single car made to drive round the circuit at constant speed. With the controller attempting to track the car, a sequence of 25 images was captured. The location of the car was then determined by eye in each image, and again the offset of this position from the optical axis was used to gauge the performance of the tracker.

In addition to these objective measurements, a subjective assessment of each of the trackers was also carried out.

Results

Table 1 shows the results from the real-world evaluations.

Table 1. Comparison of real-world performances of evolved and hand-made light and motion tracking controllers. Mean tracking errors are in degrees.

Controller	Mean Error
Evolved light-tracker	5.09 ± 1.49
Hand-made light-tracker	1.41 ± 0.15
Evolved motion-tracker	11.8 ± 1.1
Hand-made motion-tracker	11.0 ± 0.9

As can be seen, the hand-made controllers do better than the evolved controllers at both light-tracking and motion-tracking, although in the case of the motion-tracker the difference is not significant. Subjectively however, the evolved light-tracker seems to perform as well as the hand-made one. In fact, the score for the hand-made controller is a bit misleading here since the algorithm used to locate the light for the purposes of evaluation is the same as the one the program uses to track the light! Conversely, the evolved motion-tracker seems to do as well as the hand-made program on the objective assessment, but in fact the resulting controller is substantially less robust than the hand-made one. In particular, as soon as the real-world testing environment begins to diverge from the simulated environment, e.g. with bigger targets, the evolved motion-tracker goes badly wrong. The hand-made motion-tracker on the other hand was designed to cope with any size target and so has no problems here.

5 Related Work

As discussed at the start of paper, several other researchers use the term 'shaping', principally Dorigo.[2,4] There are many similarities between our work and his — a multi-agent architecture, some sort of staged learning and an evolutionary learning system — and our work can be seen as an extension to reduce further the required human design input.

Ulrich Nehmzow[25] uses the word 'shaping' somewhat differently to refer to external reinforcement provided by a human rather than by some automatic reinforcement program.

Several other researchers have looked at using multiple GP trees in concert to improve performance on complicated tasks,[26,27] but not in a shaping context as far as we know.

There has also been some work on getting GP to learn visual processing,[28,29] but we are not aware of any work using it for visual tracking tasks.

Jakobi[30] has evolved a neural network controller to perform 1-D motion tracking on a real robot. He also uses a noisy simulator to get effective simulation to reality transfer, but performs substantially more pre-processing than I do. In Jakobi's tracking task, only a narrow horizontal strip of the image is examined, and this strip is further quantized to reduce it to a one dimensional 32 pixel image. These pixels are not provided directly to the controller, but are first 'frame-differenced' to highlight image motion. This differenced image is then thresholded to produce a binary 32-bit input vector with 1's in the vector showing the location of rapid image motion.

6 Conclusions and Future Work

The shaping experiments presented here show that if used carefully, shaping can significantly improve the final quality of controllers evolved for complex tasks. In fact in experiments on motion tracking that are not reported here, it was not possible to evolve a controller that worked at all without the use of shaping. However the experiments on progressive problem difficulty do indicate that if arbitration mechanisms are going to be left to the learning system, then it is important to match the shaping regime to specializations that the learning algorithm will find easy to develop.

We have also provided a further demonstration of the process of transferring a controller that has been evolved in simulation onto a real robot, using a complex visual task. At the same time, the experience with the evolved motion-tracker highlights the fact that an evolved controller will *at best* only cope with situations on which it has been trained in simulation. It would have been possible in this case to increase the 'envelope' of fitness cases presented to the evolving controllers, but this would probably have entailed testing each individual on many more fitness cases to ensure that good estimates of true fitness were obtained. Whether this is feasible for really complicated and poorly constrained tasks remains to be seen.

References

1. Jonathan H. Connell and Sridhar Mahadevan. Rapid task learning for real robots. In Jonathan H. Connell and Sridhar Mahadevan, editors, *Robot Learning*, chapter 5, pages 105–139. Kluwer Academic Press, 1993.
2. Marco Dorigo and Marco Colombetti. Robot shaping: Developing situated agents through learning. Technical Report TR-92-040, International Computer Science Institute, Berkley, CA 94704, April 1993.
3. Wei-Po Lee, John Hallam, and Henrik Lund. Applying genetic programming to evolve behaviour primitives and aribtrators for mobile robots. In *Proc. 4th Int. Conf. on Evolutionary Computation*. IEEE Press, 1997.
4. Marco Dorigo and M. Colombetti. *Robot Shaping: An Experiment in Behaviour Engineering*. MIT Press/Bradford Books, 1998.
5. B.F. Skinner. *The Behaviour of Organisms: An Experimental Analysis*. D. Appleton Century, 1938.
6. Henrik Lund and Orazio Miglino. From simulated to real robots. In *Proc 3rd IEEE Conf. on Evolutionary Computation*. IEEE Press, 1996.
7. Nick Jakobi. Evolutionary robotics and the radical envelope of noise hypothesis. *Adaptive Behavior*, 6:325–368, 1997.

8. Orazio Miglino, Henrik Lund, and Stefano Nolfi. Evolving mobile robots in simulated and real environments. *Artifical Life*, 2(4):417–434, 1995.
9. J.H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, 1975.
10. D.H. Ackley and M.L. Littman. Generalization and scaling in reinforcement learning. In D.S. Touretsky, editor, *Advances in Neural Information Processing Systems 2.* Morgan Kaufmann, San Mateo, CA, 1990.
11. C.J.C.H. Watkins. *Learning with Delayed Rewards.* PhD thesis, Psychology Department, University of Cambridge, England, 1989.
12. Vijaykumar Gullapalli. A stochastic reinforcement learning algorithm for learning real-valued functions. *Neural Networks*, 3:671–692, 1990.
13. Juan Carlos Santamaria, Richard Sutton, and Ashwin Ram. Experiments with reinforcement learning in problems with continuous state and action spaces. Technical Report 96-088, COINS, Univ. of Massachusetts, Amherst, MA, 1996.
14. John R. Koza. *Genetic programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, 1992.
15. Simon Perkins. *Incremental Acquisition of Complex Visual Behaviour using Genetic Programming and Robot Shaping.* PhD thesis, Dept. of Artificial Intelligence, University of Edinburgh, 5 Forrest Hill, Edinburgh, Scotland, 1999. Available from `http://www.dai.ed.ac.uk/students/simonpe/pubs.html`.
16. Melanie Mitchell. *An Introduction to Genetic Algorithms.* MIT Press, Cambridge, MA, 1996.
17. Wolfgang Banzhaf, Peter Nordin, Robert E. Keller, and Frank D. Francone. *Genetic Programming: An Introduction.* Morgan Kaufmann, San Francisco, CA, 1998.
18. Riccardo Poli. Parallel distributed genetic programming. Technical Report CSRP-96-15, School of Computer Science, University of Birmingham, UK, September 1996.
19. Astro Teller and Manuela Veloso. PADO: Learning tree-structured algorithms for orchestration into an object recognition system. Technical report, Department of Computer Science, CMU, Pittsburgh, PA, 1996.
20. Astro Teller and David Andre. Automatically choosing the number of fitness cases: The rational allocation of trials. In Koza et al., editor, *Proc. Genetic Programming '97.* Morgan Kaufmann, 1997.
21. Sridhar Mahadevan and Jonathan Connell. Automatic programming of behaviour-based robots using reinforcement learning. *Artificial Intelligence*, 55:311–365, 1992.
22. Minoru Asada, Shoichi Noda, Sukoya Tawaratsumida, and Koh Hosoda.

Purposive behaviour acquisition for a real robot by vision-based reinforcement learning. *Machine Learning*, 23:279–303, 1996.
23. Simon Perkins and Gillian Hayes. Robot shaping — principles, methods and architectures. Technical Report 795, Dept. of Artificial Intelligence, Univ. of Edinburgh, Scotland, 1996.
24. Paul Cohen. *Empirical Methods for Artificial Intelligence*. MIT Press, 1995.
25. Ulrich Nehmzow and Brendan McGonigle. Achieving rapid adaptations in robots by means of external tuition. In D. Cliff, J.-A. Meyer, and S. Wilson, editors, *From Animals to Animats 3: Proc. 3rd Int. Conf. Simulation of Adaptive Behavior*. MIT Press, March 1994.
26. Sean Luke and Lee Spector. Evolving teamwork and coordination with genetic programming. In *Genetic Programming '96*, Stanford University, 1996.
27. Thomas Haynes and Sandip Sen. Crossover operators for evolving a team. In *Genetic programming '97*. Morgan Kaufmann, 1997.
28. Ricardo Poli. Genetic programming for image analysis. In *Genetic Programming '96*. Morgan Kaufmann, 1996.
29. Walter Tackett. Genetic programming for feature discovery and image discrimination. In *Proc. 5th Int. Conf. on Genetic Algorithms*, 1993.
30. Nick Jakobi. Evolving motion-tracking behaviour for a panning camera head. In R. Pfeifer, B. Blumberg, J-A. Meyer, and S. Wilson, editors, *From Animals to Animats 5: Proc. 5th Int. Conf. Simulation of Adaptive Behavior*. MIT Press, 1998.